Pi-Space Software Tutorial

Table of Contents

Introduction

This document describes the steps required to build a software project using the Pi-Space Physics Theory and the Pi-Space Core Library. Here I will show how the core library works and the tasks that it can accomplish. There are a couple of steps to understand the Pi-Space Physics Theory. The first is to understand the Physics Theory and its concepts. To do this please visit the Pi-Space Physics Theory. If one just prefers to understand just the formulas, then please download the Open Source Source Forge project and take a look at the Formulas Only Document which is on the Pi-Space Theory site.

Specifically

The Core library is written in Java and can be downloaded from Source Forge.

Source Code

http://sourceforge.net/projects/pispacephysicssoftware/

Documentation

http://www.pispacephysicstheory.com/

Code And Source

Latest code can be found as a jar

pispace.jar

The source code can be found in a zip file

pispace_src.zip

Source Code Repository (git repository)

git clone git://git.code.sf.net/p/pispacephysicssoftware/code pispacephysicssoftware-code

Chosen Development Platform

Eclipse (Import as this type which includes dependencies in the repository)

Creating Your Own App

Sample App to test Kinetic Energy

```
        @Test
        public void testKE() {

                //Units Metres, KE Pi-Space and Newton
                getLogger().debug("****** Units Metres, KE Pi-Space and Newton");

                //Test Velocity
                double mph = 3600.0;

                PiSpaceFormulas pse = new PiSpaceFormulas(
                                PiSpaceFormulas.UNIT_METRES_PER_HOUR);

                getLogger().debug("Velocity = " + mph + " Meters Per Hour");
                getLogger().debug("Newtonian KE =" +
pse.getNewtonianKEFromVelocity(mph));
                getLogger().debug("Pi-Space KE =" +
pse.getPiSpaceKEUseNewtonianUnits(mph));
                getLogger().debug("");

                double accuracy = 0.1;

                assertEquals("Passed",
                                pse.getNewtonianKEFromVelocity(mph),
                                pse.getPiSpaceKEUseNewtonianUnits(mph),
                                accuracy);
        }
```

Understanding the Units of Distance and Time

The Pi-Space Formulas Core library contains a set of Unit Types one of which can be called in the constructor of PiSpaceFormulas class. The PiSpaceFormulas class contains the main Pi-Space methods

and their classic equivalents.

The Units are

```
public static final int UNIT_METRES_PER_HOUR = 1;
public static final int UNIT_METRES_PER_SECOND = 2;

public static final int UNIT_KILOMETRES_PER_HOUR = 3;
public static final int UNIT_KILOMETRES_PER_SECOND = 4;

public static final int UNIT_MILES_PER_HOUR = 5;
public static final int UNIT_FEET_PER_HOUR = 6;
public static final int UNIT_FEET_PER_SECOND = 7;
```

Classic API versus Relativistic API

Einstein defined units in terms of fractions of the Speed of Light. Newton in his work did not need to factor in the Speed of Light. In the API, there are methods which take Velocities which are a fraction of the Speed of Light and Classic one which do not. Typically the method name is prefixed with Newtonian for the Classic work.

e.g.

```
Naming convention of the functions

getPiSpace*() - pure Pi-Space Function using Fraction of Speed of Light
getPiSpace*UseNewtonianUnits() - Pi-Space Function using Newtonian Units

getNewtonian/Einstein/Pitot...*() - Traditional Physics Formulas
```

More specifically

```
/*
 *
 * Pi-Space Kinetic Energy
 *
 * input per hour e.g. 20 MPH
 *
 * @param newtonianVelocity
 * @return
```

```
         */
        public double getPiSpaceKEUseNewtonianUnits(double newtonianVelocity)...
```

Versus

```
        /*
         *
         * Pi-Space Kinetic Energy
         *
         * input fraction speed of light
         *
         * Note: If we're using an energy formula, we need to convert velocity into
         * this form.  For example, calculating the Final Velocity Newtonian
analogue.
         *
         * @param velocityFractSpeedLight
         * @return
         */
        public double getPiSpaceKE(double velocityFractSpeedLight) ...
```

Importing As An Eclipse Project And Running The Code

The project can also be checked out via 'git clone' and then imported into Eclipse (or other preferred ide).

To understand the range of functions on offer, please visit the junit tests associated with the core library which test and demonstrate the ranges and types of calculations that the core library can calculate.

See source code location within project

src/pispace/core/test/

Sample tests are

TestPiSpaceNavierStokes.java

TestPiSpaceKineticEnergy.java

TestPiSpaceHarmonicOscillator.java

Using The Orbits Mechanism

The idea behind the Orbit Mechanism is that one provides Gravity and the position in that field relative to the Center of Gravity. Then one adds the mass and the velocity and the angle with respect to Gravity and one just calls the method over and over again until one gets a solution to a problem. Each step is measured with respect to time t which can be expressed in milliseconds. The idea is that this is a simple but powerful method to calculate the next position in a trajectory where one can also add criteria. This could be useful for a Video Game or such like.

See

TestPiSpaceTrajectory.java

```
@Test
public void testABallisticTrajectoryEngineWithTimeCriteria() {

	PiSpaceFormulas pse = new PiSpaceFormulas(
			PiSpaceFormulas.UNIT_METRES_PER_SECOND);

	getLogger().debug("15 degrees relative to horizon, use Pi-Space Formulas");

	double mass1 = 100;

	double timems = 1000;

	double massOfEarth = 5.98E24;
	double radiusOfEarth = 6.378E7;
	double distanceFromCOG = radiusOfEarth;

	double angleFromHorizontalGround=15;
	double adjustedAngleFromHorizontalGround=90+angleFromHorizontalGround;

	PiSpaceObjectTrajectoryInfo t = new PiSpaceObjectTrajectoryInfo(
			0, distanceFromCOG, //x1,y1 position
			0, 0,  //x2,y2 cog
			75.7, //velocity
			adjustedAngleFromHorizontalGround, //direction of movement
			timems, //milliseconds
			radiusOfEarth,  //radius of planet
			mass1,  //mass of object
			massOfEarth, //mass of earth
			true //use Pi-Space formulas
			);

	ArrayList pvs = new ArrayList();
```

```
            PiSpaceMaxTimeCriteria pv = new PiSpaceMaxTimeCriteria(3000);
            pvs.add(pv);
            t.setCriterias(pvs);

            TrajectoryEngine te = new TrajectoryEngine();
            te.runEngine(t,PiSpaceFormulas.UNIT_METRES_PER_SECOND);


      }
```

Here we define a planet with a radius and mass and the mass of the object in question. We choose time of 1 second (1000 milliseconds) so each steps is 1 second. We define a criteria of 3 seconds and pass this into the TrajectoryEngine which will stop when the criteria is met so it will run for 3 seconds.

Running it...

```
[main] DEBUG pispace.core.PiSpaceFormulas  - ***Construct PiSpaceFormula with Unit Id = 2
UNIT_METRES_PER_SECOND c()=2.99792458E8 cSquared()=8.987551787368176E16
2    [main] DEBUG pispace.core.test.TestPiSpaceTrajectory  - 15 degrees relative to
horizon, use Pi-Space Formulas
54   [main] DEBUG pispace.core.PiSpaceFormulas  - ***Construct PiSpaceFormula with Unit Id
= 2 UNIT_METRES_PER_SECOND c()=2.99792458E8 cSquared()=8.987551787368176E16
55   [main] DEBUG pispace.core.PiSpaceFormulas  -
------------------------------------------------------------- ITERATION START
55   [main] DEBUG pispace.core.PiSpaceFormulas  - Object initial velocity is 75.7
55   [main] DEBUG pispace.core.PiSpaceFormulas  - Milliseconds is 1000.0
55   [main] DEBUG pispace.core.PiSpaceFormulas  - Time milliseconds/1000 is 1.0
55   [main] DEBUG pispace.core.PiSpaceFormulas  - AngleAxesOffset from field is 0.0
.....

.....

121  [main] DEBUG pispace.core.PiSpaceFormulas  -
122  [main] DEBUG pispace.core.PiSpaceFormulas  - Fraction speed of light is
2.4434580286275916E-5
122  [main] DEBUG pispace.core.PiSpaceFormulas  -
122  [main] DEBUG pispace.core.PiSpaceFormulas  - Delta velocity is SPEEDS.UP NEGATIVE
VELOCITY -9.809643694048248(+speeds.up -slows.down) (new:-9.836329500417033-old:-
0.026685806368785592)
122  [main] DEBUG pispace.core.PiSpaceFormulas  -
------------------------------------------------------------- ITERATION END, CHECK
CRITERIAS
122  [main] DEBUG pispace.core.trajectory.criteria.PiSpaceMaxTimeCriteria  - Max time
criteria is 3000.0  timeMs is 3000.0
122  [main] DEBUG pispace.core.trajectory.criteria.PiSpaceMaxTimeCriteria  - !!!!!!!!!
Time criteria met at  time t = 3000.0 and distance 217.0001396317371
122  [main] DEBUG pispace.core.PiSpaceFormulas  -
------------------------------------------------------------- ITERATION END, CHECK
OBJECTS
```

There is code for Velocities near the speed of light and at Classical Newtonian velocities.

Using The Energy Manager

The Energy Manager is how the Pi-Space Theory adds up effects on mass. This is the way that Navier Stokes is treated in the Theory. We add up all the effects on the mass in the sample liquid and can then solve for either Energy or Velocity.

See

TestPiSpaceEnergyManagerPitotPressure

```
	@Test
	public void testSolveForVelocity() {

		// Pitot Pressure
		getLogger().debug("");
		getLogger().debug("****** Pitot Pressure");

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		double dynamicPressure = 1.040; // lb/ft^2
		double density = 0.002297; // slug/ft^3

		double expectedVelocity = pse.getPitotVelocityFromDynamicPressure(
				dynamicPressure, density);

		getLogger().debug("");
		getLogger()
				.debug("Pitot Newtonian pressure to velocity for dyn pressure= "
						+ dynamicPressure
						+ " density= "
						+ density
						+ " to velocity "
						+ expectedVelocity );

		PiShellEnergyManager peng = new PiShellEnergyManager(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		HashMap<String,Double> piShellEnergyChange = new HashMap();
		piShellEnergyChange.put("Pitot", new Double(dynamicPressure/density));

		peng.setEnergyChangesNewtonian(piShellEnergyChange);

		double velocity = peng.getNewtonianVelocity();
```

```
            getLogger().debug("");
            getLogger().debug("Derived Energy manager velocity is "+velocity);
            getLogger().debug("Expected Velocity is "+expectedVelocity);

            getLogger().debug("");

            double accuracy = 0.1;  //In theory, Pi-Space Energy Manager
                                                    //is more accurate as it does not use
ArcSin

            assertEquals("Passed",
                        expectedVelocity,
                        velocity,
                        accuracy);
      }
```

```
0    [main] DEBUG pispace.core.test.TestPiSpaceEnergyManagerPitotPressure  -
1    [main] DEBUG pispace.core.test.TestPiSpaceEnergyManagerPitotPressure  - ****** Pitot
Pressure
3    [main] DEBUG pispace.core.PiSpaceFormulas  - ***Construct PiSpaceFormula with Unit Id
= 2 UNIT_METRES_PER_SECOND c()=2.99792458E8 cSquared()=8.987551787368176E16
3    [main] DEBUG pispace.core.test.TestPiSpaceEnergyManagerPitotPressure  -
3    [main] DEBUG pispace.core.test.TestPiSpaceEnergyManagerPitotPressure  - Pitot
Newtonian pressure to velocity for dyn pressure= 1.04 density= 0.002297 to velocity
30.092008088617124
4    [main] DEBUG pispace.core.PiSpaceFormulas  - ***Construct PiSpaceFormula with Unit Id
= 2 UNIT_METRES_PER_SECOND c()=2.99792458E8 cSquared()=8.987551787368176E16
4    [main] DEBUG pispace.core.manager.PiShellEnergyManager  - Pitot = 452.7644754026992
4    [main] DEBUG pispace.core.PiSpaceFormulas  - Potential Energy is 452.7644754026992
4    [main] DEBUG pispace.core.PiSpaceFormulas  - Speed of light squared is
8.987551787368176E16
4    [main] DEBUG pispace.core.PiSpaceFormulas  - Pi-Space Area val is 5.037684189359004E-
15
4    [main] DEBUG pispace.core.test.TestPiSpaceEnergyManagerPitotPressure  -
4    [main] DEBUG pispace.core.test.TestPiSpaceEnergyManagerPitotPressure  - Derived
Energy manager velocity is 30.092008088617085
4    [main] DEBUG pispace.core.test.TestPiSpaceEnergyManagerPitotPressure  - Expected
Velocity is 30.092008088617124
4    [main] DEBUG pispace.core.test.TestPiSpaceEnergyManagerPitotPressure  -
```

Test Velocity And Angles

```
public void testAngle() {

            double xvelocity = 10;
            double yvelocity = 20;

            PiSpaceFormulas psf = new
PiSpaceFormulas(PiSpaceFormulas.UNIT_METRES_PER_HOUR);
            AngleForVelocities ta = new
AngleForVelocities(xvelocity,yvelocity,psf);
            getLogger().debug("x ="+ta.getxVelocity());
            getLogger().debug("y ="+ta.getyVelocity());
            getLogger().debug("xy ="+ta.getXyVelocity());
            getLogger().debug("angle ="+ta.getAngle());
      }
```

Test BigDecimal Trignometry

```
public void testAsin() {

            BigDecimal bd1 = new BigDecimal(1.0);
            //BigDecimal bd2 = new
BigDecimal(0.000000000000000000055632502802680929);
            BigDecimal bd2 = new BigDecimal(0.055632502802680929);
            BigDecimal bd3 = bd1.subtract(bd2);

            //System.out.println("arcSin(1.0 -
0.000000000000000000055632502802680929)="+BigDecimalTrignometryService.arcSin(0.9));
            //System.out.println("arcSin(1.0 -
0.000000000000000000055632502802680929)="+BigDecimalTrignometryService.arcSin(t));


            BigDecimal bd = BigDecimalTrignometryService.asin(bd3);
            getLogger().debug("arcSin(1.0 -
0.000000000000000000055632502802680929)="+bd.doubleValue());


             getLogger().debug("");

      }
```

Test Law of the Cosines

```
@Test
public void test() {

	PiSpaceFormulas pse = new PiSpaceFormulas(
			PiSpaceFormulas.UNIT_METRES_PER_HOUR);

	getLogger().debug("Test Law of Cosines");

	double v = 0.1;
	double a = 0.5*0.1*1.0*1.0;

	double val = pse.getLawOfCosinesDistance(v, a, 180);

	getLogger().debug("Val is "+val);
	getLogger().debug("");

	val = pse.getLawOfCosinesDistance(val, a, 180);

	getLogger().debug("Val is "+val);

}
```

Test Law of the Sines

```
@Test
public void test() {

	PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_HOUR);

	getLogger().debug("Test Law of Sines");

	double v1 = 20;
	double angle = 90;
	double v2 = 10;

	double a = pse.getLawOfSinesAngle(v1, angle, v2);

	getLogger().debug("Val is "+a);
	getLogger().debug("");

}
```

Test Average Velocity

```
	@Test
	public void test() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_HOUR);

		getLogger().debug("");
		getLogger().debug("****** Test Average Velocity (Pi-Space and Newton) 
0.0 to 1.0 C");

		double startVelocity = 0.0;
		double endVelocity = 0.1;

		getLogger().debug("");
		getLogger().debug("Newton startVelocity= "
				+ startVelocity
				+ " endVelocity= "
				+ endVelocity
				+ " gives value= "
				+ (pse.getNewtonianGetAverageVelocity(startVelocity,
						endVelocity)));

		getLogger().debug("");
		getLogger().debug("Pi-Space startVelocity= " + startVelocity
				+ " endVelocity= " + endVelocity + " gives value= "
				+ (pse.getPiSpaceAverageVelocity(startVelocity, 
endVelocity)));

		double accuracy = 0.1;

		assertEquals("Passed",

	pse.getNewtonianGetAverageVelocity(startVelocity,
					endVelocity),
					pse.getPiSpaceAverageVelocity(startVelocity, 
endVelocity),
				accuracy);
	}
```

Test Radius Excess

```
	@Test
	public void testSolarSystemPlanets() {

		// Units Metres, Escape Velocity Pi-Space and Newton
		getLogger()
		.debug("");
		getLogger()
				.debug("****** Test Radius Excess For Solar System
Planets");


		// Formulas

		PiSpaceFormulas pf = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		// Calculate the Escape Velocity for each planet

		Planets planets = new Planets();

		Iterator it = planets.getPlanets().entrySet().iterator();
		while (it.hasNext()) {
			Map.Entry pairs = (Map.Entry) it.next();
			// getLogger().debug(pairs.getKey() + " = " + pairs.getValue());

			Planet p = (Planet) pairs.getValue();

			double radiusExcessEinstein = pf.getEinsteinRadiusExcess(
					p.getMass());

			double radiusExcessPiSpace = pf.getPiSpaceRadiusExcess(
					p.getMass());

			getLogger().debug("Name of planet " + pairs.getKey());
			getLogger().debug("Mass of planet " + p.getMass());
			getLogger().debug("Radius of planet " + p.getRadius());
			getLogger().debug("Radius Excess Einstein (metres) " +
radiusExcessEinstein);
			getLogger().debug("Radius Excess PiSpace (metres) " +
radiusExcessPiSpace);


			getLogger().debug("");

			double accuracy = 1.0;

			assertEquals("Passed",
					radiusExcessEinstein*3.0, //difference between two
approaches
					radiusExcessPiSpace,
					accuracy);
		}

	}
```

Test Velocity To Energy And Back Again

```
	@Test
	public void testVelocityToEnergyAndBackAgain() {

		// Convert Velocity to Energy and Back to Velocity Again

		getLogger()
				.debug("****** Velocity to Energy and Back");

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		double velocity = 20000;
		double velocityEnergy = 0.5*(velocity*velocity);

		getLogger().debug("Velocity " + velocity);
		getLogger().debug("Velocity to Energy " + velocityEnergy);


		getLogger().debug("Energy And Back Again "
				+
pse.getPiSpacePEtoVelocityUseNewtonianUnits(velocityEnergy));

		double accuracy = 0.1;

		assertEquals("Passed",

	pse.getPiSpacePEtoVelocityUseNewtonianUnits(velocityEnergy),
				velocity,
				accuracy);
	}
```

Test Curvature

```
	@Test
	public void testCurvatureLargeVelocity() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		getLogger().debug("Curvature for 90 degrees straight up, use Pi-Space
Formulas");

		double particleVelocity = pse.getC();
		double paticleDist = pse.getC();
		double accelerationDist = 9.8;
		double velocityAngle = 90.0;

		double angleAdjustment =
pse.getPiSpaceTrajectoryVectorOffsetDueToLargeVelocityAndGravity(
				particleVelocity,
				paticleDist,
				accelerationDist,
				velocityAngle
				);

		getLogger().debug("Angle Adjustment is "+angleAdjustment);

	}
```

Test De Broglie

```
	@Test
	public void testFractionCMomentum() {

		//De Broglie
		PiSpaceFormulas pse = new
PiSpaceFormulas(PiSpaceFormulas.UNIT_METRES_PER_HOUR);
		getLogger().debug("");

		getLogger().debug("****** De Broglie Original");
		getLogger().debug(
				"Mass 1.0 at 0.1C is "
					+ pse.getDeBroglieWaveMomentum(0.1, 1.0));

		//De Broglie Pi-Space
		pse = new PiSpaceFormulas(PiSpaceFormulas.UNIT_METRES_PER_HOUR);
		getLogger().debug("");

		getLogger().debug("****** De Broglie Pi-Space");
		getLogger().debug(
				"0.1C is "
					+ pse.getPiSpaceDeBroglieWaveMomentum(0.1,
1.0));

		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getDeBroglieWaveMomentum(0.1, 1.0),
				pse.getPiSpaceDeBroglieWaveMomentum(0.1, 1.0),
				accuracy);
	}
```

Test Distance

```
	@Test
	public void test2() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_HOUR);

		getLogger().debug("");
		getLogger()
				.debug("****** Test Distance (Pi-Space and Newton) use
gamma");

		double velocityFractionOfLight=0.0001;
		double accelerationFractionOfLightSquared=0.01;
		double time=10;


		getLogger().debug("");
		getLogger().debug("Newton startVelocity= "
				+ velocityFractionOfLight
				+ " accelerationFractionOfLightSquared= "
				+ accelerationFractionOfLightSquared
				+ " gives value= "
				+ (pse.getNewtonianDistance(
						velocityFractionOfLight,
						accelerationFractionOfLightSquared,
						time)));


		getLogger().debug("");
		getLogger().debug("Pi-Space startVelocity= "
				+ velocityFractionOfLight
				+ " accelerationFractionOfLightSquared= "
				+ accelerationFractionOfLightSquared
				+ " gives value= "
				+ (pse.getPiSpaceDistance(
						velocityFractionOfLight,
						accelerationFractionOfLightSquared,
						time,
						true)));


		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getNewtonianDistance(
						velocityFractionOfLight,
						accelerationFractionOfLightSquared,
						time),
				pse.getPiSpaceDistance(
						velocityFractionOfLight,
						accelerationFractionOfLightSquared,
						time,
						true),
				accuracy);
```

```
}
```

Test Distance Under Gravity

```
	@Test
	public void test() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		getLogger().debug("");
		getLogger()
				.debug("****** Test Upward Distance Under Gravity (Pi-Space
and Newton) use Newtonian interface for Pi-Space");

		double velocityMPH=100.0; //drop
		double accelerationMSS=9.8; //gravity


		getLogger().debug("");
		getLogger().debug("Newton startVelocity= "
				+ velocityMPH
				+ " accelerationFractionOfLightSquared= "
				+ accelerationMSS
				+ " gives value= "
				+ (pse.getNewtonianDistanceTraveledUnderGravity(
						velocityMPH,
						accelerationMSS)));


		getLogger().debug("");
		getLogger().debug("Pi-Space startVelocity= "
				+ velocityMPH
				+ " accelerationFractionOfLightSquared= "
				+ accelerationMSS
				+ " gives value= "
				+
(pse.getPiSpaceDistanceTraveledUnderGravityUseNewtonianUnits(
						velocityMPH,
						accelerationMSS)));

		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getNewtonianDistanceTraveledUnderGravity(
						velocityMPH,
						accelerationMSS),

	pse.getPiSpaceDistanceTraveledUnderGravityUseNewtonianUnits(
						velocityMPH,
						accelerationMSS),
				accuracy);
	}
```

Test Elastic Collision

```
	@Test
	public void
testV1FinalPiSpaceAndNewtonLargeVelocity1StationaryV2AndShowV2LargeVelocity2() {

		//This is a high speed collision relative to particle physics
		//Newtonian formula fails here

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		getLogger().debug("");
		getLogger()
				.debug("****** Test Elastic Collision (Pi-Space and Newton)
use Newtonian Units for Pi-Space");


		double mass1=3.0;
		double velocity1=-pse.getC()*0.7;
		double mass2=500.0;
		double velocity2=pse.getC()*0.8;


		getLogger().debug("");
		getLogger().debug(
				"Newton mass1= "
				+ mass1
				+ " velocity1= "
				+ velocity1
				+ " mass2= "
				+ mass2
				+ " velocity2= "
				+ velocity2
				+ " elastic collision gives value newton v1= "
				+ (pse.

	getNewtonianVelocityV1FinalFromElasticCollisionMomentumPairsV1V2(
					mass1, velocity1, mass2, velocity2))
				+ " elastic collision gives value pispace v1= "
				+ (pse.


getPiSpaceVelocityV1FinalFromElasticCollisionMomentumPairsV1V2UseNewtonianUnits(
					mass1, velocity1, mass2, velocity2))
				+ " elastic collision gives value pispace v2= "
				+ (pse.


getPiSpaceVelocityV2FinalFromElasticCollisionMomentumPairsV1V2UseNewtonianUnits(
					mass1, velocity1, mass2,
velocity2))
				);


		double accuracy = 0.1;
```

```
            assertEquals("Passed",
                        pse.

getPiSpaceVelocityV1FinalFromElasticCollisionMomentumPairsV1V2UseNewtonianUnits(
                              //getNewtonianVelocityV1FinalFromElasticCollisionMome
ntumPairsV1V2(
                              mass1, velocity1, mass2, velocity2),
                        pse.


getPiSpaceVelocityV1FinalFromElasticCollisionMomentumPairsV1V2UseNewtonianUnits(
                              mass1, velocity1, mass2, velocity2),
                        accuracy);
      }
```

Test Elastic Collision At An Angle

```
	@Test
	public void testCollisionAtAnAngleBetweenTwoSpheresUsePiSpace3() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		getLogger().debug("");
		getLogger().debug(
				"****** Test Collision at an angle between two spheres use
pi-space");

		//Before collision
		PiSpaceSphere s1 = new PiSpaceSphere();
		s1.setMass(3.0);
		s1.setxVelocity(1.0);
		s1.setyVelocity(1.0);
		s1.setXpos(1.0);
		s1.setYpos(1.0);
		s1.setDiameter(1.0);

		PiSpaceSphere s2 = new PiSpaceSphere();
		s2.setMass(3.0);
		s2.setxVelocity(-1.0);
		s2.setyVelocity(-1.0);
		s2.setXpos(2.0);
		s2.setYpos(1.0);
		s2.setDiameter(1.0);

		//After collision
		PiSpaceSphere s3 = new PiSpaceSphere(s1);
		PiSpaceSphere s4 = new PiSpaceSphere(s2);

		boolean useClassic = false;
		pse.getPiSpaceResultFromSphereCollisionAtAnAngleUseNewtonianUnits(
				s1, s2, s3,s4,useClassic);

		getLogger().debug("");
		getLogger()
				.debug("Newton mass1= "
						+ s1.getMass()
						+ " xvelocity1= "
						+ s1.getxVelocity()
						+ " yvelocity1= "
						+ s1.getyVelocity()
						+ " mass2= "

						+ s2.getMass()
						+ " xvelocity2= "
						+ s2.getxVelocity()
						+ " yvelocity2= "
						+ s2.getyVelocity());

		getLogger()
				.debug("Newton mass3= "
```

```
                                + s3.getMass()
                                + " xvelocity3= "
                                + s3.getxVelocity()
                                + " yvelocity3= "
                                + s3.getyVelocity()

                                + " mass4= "
                                + s4.getMass()
                                + " xvelocity4= "
                                + s4.getxVelocity()
                                + " yvelocity4= "
                                + s4.getyVelocity()
                                );

        double accuracy = 0.1;

        PiSpaceSphere s4Result = new PiSpaceSphere();
        s4Result.setMass(3.0);
        s4Result.setxVelocity(1.0);
        s4Result.setyVelocity(1.0);
        s4Result.setXpos(2.0);
        s4Result.setYpos(2.0);
        s4Result.setDiameter(1.0);

        assertEquals(
                    "Passed",
                    s4.getxVelocity(),
                    s4Result.getxVelocity(),
                    accuracy);
}
```

Test Pitot Pressure

```
	@Test
	public void testSolveForVelocity() {

		// Pitot Pressure
		getLogger().debug("");
		getLogger().debug("****** Pitot Pressure");

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		double dynamicPressure = 1.040; // lb/ft^2
		double density = 0.002297; // slug/ft^3

		double expectedVelocity = pse.getPitotVelocityFromDynamicPressure(
				dynamicPressure, density);

		getLogger().debug("");
		getLogger()
				.debug("Pitot Newtonian pressure to velocity for dyn
pressure= "
					+ dynamicPressure
					+ " density= "
					+ density
					+ " to velocity "
					+ expectedVelocity );

		PiShellEnergyManager peng = new PiShellEnergyManager(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		HashMap<String,Double> piShellEnergyChange = new HashMap();
		piShellEnergyChange.put("Pitot", new Double(dynamicPressure/density));

		peng.setEnergyChangesNewtonian(piShellEnergyChange);

		double velocity = peng.getNewtonianVelocity();

		getLogger().debug("");
		getLogger().debug("Derived Energy manager velocity is "+velocity);
		getLogger().debug("Expected Velocity is "+expectedVelocity);

		getLogger().debug("");

		double accuracy = 0.1;  //In theory, Pi-Space Energy Manager
					//is more accurate as it does not
use ArcSin

		assertEquals("Passed",
				expectedVelocity,
				velocity,
				accuracy);
	}
```

Test Potential Energy

```
	@Test
	public void testSolveForVelocity() {

		//TODO - This is not yet done.

		//Test a FluidFlowArea

		PiShellEnergyManager peng = new PiShellEnergyManager(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		getLogger()
				.debug("****** Units Metres, Velocity Pi-Space");

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		double EarthMass = 5.98E24;
		double EarthRadius = 6.378E6;

		double massOfPlanet = EarthMass;
		double radiusOfPlanet = EarthRadius;

		double gpe = pse.getNewtonianGravitationalPotentialEnergy(massOfPlanet,
				radiusOfPlanet);

		double expectedVelocity =
pse.getPiSpacePEtoVelocityUseNewtonianUnits(gpe);

		getLogger().debug("Mass of planet " + massOfPlanet);
		getLogger().debug("Radius of planet " + radiusOfPlanet);
		getLogger().debug("Gravitation Potential Energy " + gpe);

		getLogger().debug("Newton Velocity "
				+ pse.getNewtonianPeToVelocity(gpe));
		getLogger().debug("Pi-Space Escape Velocity "
				+ expectedVelocity);

		HashMap<String,Double> piShellEnergyChange = new HashMap();
		piShellEnergyChange.put("Potential Energy", new Double(gpe));

		peng.setEnergyChangesNewtonian(piShellEnergyChange);

		double velocity = peng.getNewtonianVelocity();

		getLogger().debug("");
		getLogger().debug("Derived Energy manager velocity is "+velocity);
		getLogger().debug("Expected Velocity is "+expectedVelocity);

		getLogger().debug("");

		double accuracy = 15.0;  //In theory, Pi-Space Energy Manager
					//is more accurate as it does not
use ArcSin
```

```
        assertEquals("Passed",
                     expectedVelocity,
                     velocity,
                     accuracy);
    }
```

Test Escape Velocity

```
	@Test
	public void test() {

		// Units Metres, Escape Velocity Pi-Space and Newton
		getLogger()
				.debug("****** Units Metres, Escape Velocity Pi-Space and
Newton");

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		double EarthMass = 5.98E24;
		double EarthRadius = 6.378E6;

		double massOfPlanet = EarthMass;
		double radiusOfPlanet = EarthRadius;

		double gpe = pse.getNewtonianGravitationalPotentialEnergy(massOfPlanet,
				radiusOfPlanet);

		getLogger().debug("Mass of planet " + massOfPlanet);
		getLogger().debug("Radius of planet " + radiusOfPlanet);
		getLogger().debug("Gravitation Potential Energy " + gpe);

		getLogger().debug("Newton Escape Velocity "
				+ pse.getNewtonianPeToVelocity(gpe));
		getLogger().debug("Pi-Space Escape Velocity "
				+ pse.getPiSpacePEtoVelocityUseNewtonianUnits(gpe));

		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getNewtonianPeToVelocity(gpe),
				pse.getPiSpacePEtoVelocityUseNewtonianUnits(gpe),
				accuracy);
	}
```

Test Final Velocity

```
	@Test
	public void test() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		getLogger().debug("");
		getLogger()
				.debug("****** Test Final Velocity (Pi-Space and Newton) do not use gamma, use Newtonian interface for Pi-Space");

		double velocityMPH=0.0; //drop
		double accelerationMSS=9.8; //gravity

		double distance=381;  //empire state


		getLogger().debug("");
		getLogger().debug("Newton startVelocity= "
				+ velocityMPH
				+ " accelerationFractionOfLightSquared= "
				+ accelerationMSS
				+ " distance= "
				+ distance
				+ " gives value= "
				+ (pse.getNewtonianFinalVelocity(
						velocityMPH,
						accelerationMSS,
						distance)));


		getLogger().debug("");
		getLogger().debug("Pi-Space startVelocity= "
				+ velocityMPH
				+ " accelerationFractionOfLightSquared= "
				+ accelerationMSS
				+ " distance= "
				+ distance
				+ " gives value= "
				+ (pse.getPiSpaceFinalVelocityUseNewtonianUnits(
						velocityMPH,
						accelerationMSS,
						distance,
						false)));


		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getNewtonianFinalVelocity(
						velocityMPH,
						accelerationMSS,
						distance),
				pse.getPiSpaceFinalVelocityUseNewtonianUnits(
```

```
                                velocityMPH,
                                accelerationMSS,
                                distance,
                                false),
                        accuracy);
    }
```

Test Gamma Calculation

```
	@Test
	public void test() {
		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_HOUR);

		// Gamma Calculations for Relativity
		getLogger().debug("");
		getLogger().debug("****** Gamma Calculations for Relativity");

		double startVelocity = 0.1;
		double endVelocity = 0.2;

		getLogger().debug("");
		getLogger()
				.debug("Pi-Space Gamma for velocity range start velocity= "
					+ startVelocity
					+ " endVelocity= "
					+ endVelocity
					+ " gives gamma value= "
					+
pse.getPiSpaceGammaForAcceleration(startVelocity,
						endVelocity));

		startVelocity = 0.9999995;
		endVelocity = 0.9999999;

		getLogger().debug("");
		getLogger()
				.debug("Pi-Space Gamma for velocity range start velocity= "
					+ startVelocity
					+ " endVelocity= "
					+ endVelocity
					+ " gives gamma value= "
					+
pse.getPiSpaceGammaForAcceleration(startVelocity,
						endVelocity));

		// 20 MPH to 50 MPH

		double startVelocityMPH = 20.0;
		double endVelocityMPH = 50.0;
		double timeInSeconds = 1.0;

		startVelocity = pse.getFractionSpeedOfLight(startVelocityMPH);
		endVelocity = pse.getFractionSpeedOfLight(endVelocityMPH);

		getLogger().debug("");
		getLogger()
				.debug("Pi-Space Gamma for velocity range start
velocityMPH= "
					+ startVelocityMPH
					+ " endVelocityMPH= "
					+ endVelocityMPH
					+ " gives gamma value= "
```

```
                                        +
pse.getPiSpaceGammaForAcceleration(startVelocity,
                                                endVelocity));

            getLogger().debug("");
            getLogger()
                        .debug("Newtonian Acceleration for velocity range start
velocityMPH= "
                                        + startVelocityMPH
                                        + " endVelocityMPH= "
                                        + endVelocityMPH
                                        + " time in seconds "
                                        + timeInSeconds
                                        + " gives acceleration value= "
                                        +
pse.getNewtonianAcceleration(startVelocityMPH,
                                                endVelocityMPH, timeInSeconds));

            getLogger().debug("");
            getLogger()
                        .debug("Adjusted Newtonian Acceleration using gamma for
velocity range start velocityMPH= "
                                        + startVelocityMPH
                                        + " endVelocityMPH= "
                                        + endVelocityMPH
                                        + " time in seconds "
                                        + timeInSeconds
                                        + " gives acceleration value= "
                                        +
(pse.getNewtonianAcceleration(startVelocityMPH,
                                                endVelocityMPH, timeInSeconds))
                                        *
pse.getPiSpaceGammaForAcceleration(startVelocity,
                                                endVelocity));

            assertTrue("No test needed.", true);
      }
```

Test Harmonic Oscillator

```
@Test
public void test() {

	// Velocity based Newtonian Harmonic Motion
	getLogger().debug("");
	getLogger()
			.debug("****** Velocity based Newtonian Harmonic Motion");

	PiSpaceFormulas pse = new PiSpaceFormulas(
			PiSpaceFormulas.UNIT_METRES_PER_SECOND);

	double amplitude = 5.0;
	double position = 2.0;
	double springConstantk = 1; // N/s
	double mass = 2; // N/s

	getLogger().debug("");
	getLogger().debug("Pi-Space Harmonic Oscillator amplitude= "
			+ amplitude
			+ " position= "
			+ position
			+ " spring constant k= "
			+ springConstantk
			+ " mass= "
			+ mass
			+ " to velocity= "
			+ pse.getPiSpaceHarmonicOscillatorGetVelocity(amplitude,
					position, springConstantk, mass));

	getLogger().debug("");
	getLogger().debug("Newtonian Harmonic Oscillator amplitude= "
			+ amplitude
			+ " position= "
			+ position
			+ " spring constant k= "
			+ springConstantk
			+ " mass= "
			+ mass
			+ " to velocity= "
			+ pse.getNewtonianHarmonicOscillatorGetVelocity(amplitude,
					position, springConstantk, mass));

	double accuracy = 0.1;

	assertEquals("Passed",
			pse.getPiSpaceHarmonicOscillatorGetVelocity(amplitude,
					position, springConstantk, mass),
			pse.getNewtonianHarmonicOscillatorGetVelocity(amplitude,
					position, springConstantk, mass),
			accuracy);
}
```

Test Kinetic Energy

```
	@Test
	public void test1() {

		//Units Metres, KE Pi-Space and Newton
		getLogger().debug("****** Units Metres, KE Pi-Space and Newton");

		//Test Velocity
		double mph = 3600.0;

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_HOUR);

		getLogger().debug("Velocity = " + mph + " Meters Per Hour");
		getLogger().debug("Newtonian KE =" +
pse.getNewtonianKEFromVelocity(mph));
		getLogger().debug("Pi-Space KE =" +
pse.getPiSpaceKEUseNewtonianUnits(mph));
		getLogger().debug("");

		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getNewtonianKEFromVelocity(mph),
				pse.getPiSpaceKEUseNewtonianUnits(mph),
				accuracy);
	}
```

Test Length Contraction

```
	@Test
	public void testFractionCLength() {

		//Lorentz Fitzgerald Einstein
		PiSpaceFormulas pse = new
PiSpaceFormulas(PiSpaceFormulas.UNIT_METRES_PER_HOUR);
		getLogger().debug("");

		getLogger().debug("****** Length Contraction Original");
		getLogger().debug(
				"Length 1.0 at 0.1C is "
						+
pse.getEinsteinSpecialRelativityLengthContraction(1.0, 0.1));

		//Lorentz Fitzgerald Pi-Space
		pse = new PiSpaceFormulas(PiSpaceFormulas.UNIT_METRES_PER_HOUR);
		getLogger().debug("");

		getLogger().debug("****** Lorentz Fitzgerald Pi-Space");
		getLogger().debug(
				"LorentzFitzgerald 0.1C is "
						+
pse.getPiSpaceSpecialRelativityLengthContraction(1.0, 0.1));

		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getEinsteinSpecialRelativityLengthContraction(1.0,
0.1),
				pse.getPiSpaceSpecialRelativityLengthContraction(1.0, 0.1),
				accuracy);
	}
```

Test Lorentz Fitzgerald

```
	@Test
	public void test() {

		//Lorentz Fitzgerald Einstein
		PiSpaceFormulas pse = new
PiSpaceFormulas(PiSpaceFormulas.UNIT_METRES_PER_HOUR);
		getLogger().debug("");

		getLogger().debug("****** Lorentz Fitzgerald Original");
		getLogger().debug(
				"LorentzFitzgerald 0.1C is "
						+
pse.getEinsteinLorentzFitzgeraldTransformation(0.1));

		//Lorentz Fitzgerald Pi-Space
		pse = new PiSpaceFormulas(PiSpaceFormulas.UNIT_METRES_PER_HOUR);
		getLogger().debug("");

		getLogger().debug("****** Lorentz Fitzgerald Pi-Space");
		getLogger().debug(
				"LorentzFitzgerald 0.1C is "
						+
pse.getPiSpaceLorentzFitzgeraldTransformation(0.1));

		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getEinsteinLorentzFitzgeraldTransformation(0.1),
				pse.getPiSpaceLorentzFitzgeraldTransformation(0.1),
				accuracy);
	}
```

Test Navier Stokes

```
	@Test
	public void testSolveForVelocityNavStokesKineticEnergy() {

		//TODO Call Navier Stokes

		PiSpaceNavierStokes pNavStokes = new PiSpaceNavierStokes(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		//Need to do a EnergyManager version and a Navier Stokes version

		double expectedVelocity = 20;

		pNavStokes.setKineticEnergy(expectedVelocity);

		double velocity = pNavStokes.getNewtonianVelocity();

		//double expectedVelocity = getEnergyManagerEquivalent();
//getEnergyManagerEquivalent

		getLogger().debug("");
		getLogger().debug("Derived Energy manager velocity is "+velocity);
		getLogger().debug("Expected Velocity is "+expectedVelocity);

		getLogger().debug("");

		double accuracy = 15.0;  //In theory, Pi-Space Energy Manager
					//is more accurate as it does not
use ArcSin

		assertEquals("Passed",
				expectedVelocity,
				velocity,
				accuracy);
	}
```

Test Pitot Pressure

```
	@Test
	public void test() {

		//TODO Use the non ArcSin method

		// Pitot Pressure
		getLogger().debug("");
		getLogger().debug("****** Pitot Pressure");

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_FEET_PER_HOUR);

		double dynamicPressure = 1.040; // lb/ft^2
		double density = 0.002297; // slug/ft^3

		getLogger().debug("");
		getLogger()
				.debug("Pitot Newtonian pressure to velocity for dyn
pressure= "
					+ dynamicPressure
					+ " density= "
					+ density
					+ " to velocity "
					+ pse.getPitotVelocityFromDynamicPressure(
							dynamicPressure, density));

		getLogger().debug("");
		getLogger()
				.debug("Pitot Pi-Space pressure to velocity for dyn
pressure= "
					+ dynamicPressure
					+ " density= "
					+ density
					+ " to velocity "
					+
pse.getPiSpacePitotVelocityFromDynamicPressure(
							dynamicPressure, density));


		double accuracy = 1.0;

		assertEquals("Passed",
				pse.getPitotVelocityFromDynamicPressure(
						dynamicPressure, density),
					pse.getPiSpacePitotVelocityFromDynamicPressure(
						dynamicPressure, density),
				accuracy);
	}
```

Test Proper Time

```
	@Test
	public void testLightAstronaut() {

		//Proper Time

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_HOUR);

		getLogger().debug("");
		getLogger().debug("****** Proper Time Light Year");

		getLogger().debug("");
		getLogger()
				.debug("How much proper time for astronaut at 0.8C? 
Observer time 50 years.  "
					+ pse.getPiSpaceProperTime(50.0, 0.8) + " 
years.");

		assertEquals("passed",pse.getPiSpaceProperTime(50.0, 0.8), 30.0, 0.1);
	}
```

Test Radius Excess

```
	@Test
	public void testSolarSystemPlanets() {

		// Units Metres, Escape Velocity Pi-Space and Newton
		getLogger()
		.debug("");
		getLogger()
				.debug("****** Test Radius Excess For Solar System
Planets");


		// Formulas

		PiSpaceFormulas pf = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		// Calculate the Escape Velocity for each planet

		Planets planets = new Planets();

		Iterator it = planets.getPlanets().entrySet().iterator();
		while (it.hasNext()) {
			Map.Entry pairs = (Map.Entry) it.next();
			// getLogger().debug(pairs.getKey() + " = " + pairs.getValue());

			Planet p = (Planet) pairs.getValue();

			double radiusExcessEinstein = pf.getEinsteinRadiusExcess(
					p.getMass());

			double radiusExcessPiSpace = pf.getPiSpaceRadiusExcess(
					p.getMass());

			getLogger().debug("Name of planet " + pairs.getKey());
			getLogger().debug("Mass of planet " + p.getMass());
			getLogger().debug("Radius of planet " + p.getRadius());
			getLogger().debug("Radius Excess Einstein (metres) " +
radiusExcessEinstein);
			getLogger().debug("Radius Excess PiSpace (metres) " +
radiusExcessPiSpace);


			getLogger().debug("");

			double accuracy = 1.0;

			assertEquals("Passed",
					radiusExcessEinstein*3.0, //difference between two
approaches
					radiusExcessPiSpace,
					accuracy);
		}

	}
```

Test Relativistic Kinetic Energy

```
	@Test
	public void test() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_HOUR);

		// Test for Relativistic Kinetic Energy (Pi-Space and Newton)
		getLogger().debug("");
		getLogger()
				.debug("****** Test for Relativistic Kinetic Energy (Pi-
Space and Einstein)");

		double mass = 1.0;
		double velocityFractionOfLight = 0.1;

		getLogger().debug("");
		getLogger().debug("Einstein Relativistic Kinetic Energy velocityMPH= "
				+ " velocityFractionOfLight= "
				+ velocityFractionOfLight
				+ " mass "
				+ mass
				+ " gives value= "
				+ (pse.getEinsteinRelativisticKineticEnergy(
						velocityFractionOfLight, mass)));

		getLogger().debug("");
		getLogger().debug("Pi-Space Relativistic Kinetic Energy velocityMPH= "
				+ " velocityFractionOfLight= "
				+ velocityFractionOfLight
				+ " mass "
				+ mass
				+ " gives value= "
				+ (pse.getPiSpaceRelativisticKineticEnergy(
						velocityFractionOfLight, mass)));

		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getEinsteinRelativisticKineticEnergy(
						velocityFractionOfLight, mass),
				pse.getPiSpaceRelativisticKineticEnergy(
						velocityFractionOfLight, mass),
				accuracy);
	}
```

Test Schwartzchild Radius

```
	public void testSolarSystemPlanets() {

		// Units Metres, Escape Velocity Pi-Space and Newton
		getLogger()
		.debug("");
		getLogger()
				.debug("****** Test Schwarzchild Radius For Solar System
Planets");


		// Formulas

		PiSpaceFormulas pf = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		// Calculate the Escape Velocity for each planet

		Planets planets = new Planets();

		Iterator it = planets.getPlanets().entrySet().iterator();
		while (it.hasNext()) {
			Map.Entry pairs = (Map.Entry) it.next();
			// getLogger().debug(pairs.getKey() + " = " + pairs.getValue());

			Planet p = (Planet) pairs.getValue();

			double radiusExcessEinstein = pf.getEinsteinSchwarzchildRadius(
					p.getMass());

			double radiusExcessPiSpace = pf.getPiSpaceSchwarzchildRadius(
					p.getMass());

			getLogger().debug("Name of planet " + pairs.getKey());
			getLogger().debug("Mass of planet " + p.getMass());
			getLogger().debug("Normal Solar Radius of planet " +
p.getRadius());
			getLogger().debug("Schwarzchild Radius Einstein (metres) " +
radiusExcessEinstein);
			getLogger().debug("Schwarzchild Radius PiSpace (metres) " +
radiusExcessPiSpace);


			getLogger().debug("");

			double accuracy = 1.0;

			assertEquals("Passed",
					radiusExcessEinstein,
					radiusExcessPiSpace*2.0,  //difference between two
approaches
					accuracy);
		}

	}
```

Test Time Passed

```
	@Test
	public void test3() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_HOUR);

		getLogger().debug("");
		getLogger()
				.debug("****** Test Time (Pi-Space and Newton) do USE
gamma, DO NOT use Newtonian interface for Pi-Space");

		double velocityMPH=0.0;
		double accelerationMSS=0.01;
		double distance=0.405274;


		getLogger().debug("");
		getLogger().debug("Newton startVelocity= "
				+ velocityMPH
				+ " acceleration= "
				+ accelerationMSS
				+ " distance= "
				+ distance
				+ " gives value= "
				+ (pse.getNewtonianTimePassed(
						velocityMPH,
						accelerationMSS,
						distance)));


		getLogger().debug("");
		getLogger().debug("Pi-Space startVelocity= "
				+ velocityMPH
				+ " accelerationFractionOfLightSquared= "
				+ accelerationMSS
				+ " distance= "
				+ distance
				+ " gives value= "
				+ (pse.getPiSpaceTimePassed(
						velocityMPH,
						accelerationMSS,
						distance,
						true)));


		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getNewtonianTimePassed(
						velocityMPH,
						accelerationMSS,
						distance),
				pse.getPiSpaceTimePassed(
						velocityMPH,
```

```
                                accelerationMSS,
                                distance,
                                true),
                        accuracy);
    }
```

Test Transverse Kinetic Energy

```
	@Test
	public void test() {

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		// Test 8 - Velocity solution for Transverse Kinetic Energy
		getLogger().debug("");
		getLogger()
				.debug("Velocity solution for Transverse Kinetic Energy");

		double mass = 6.65E-27; // hydrogen
		double temperatureCelcius = 27.0;
		double temperatureKelvin = 273.0 + temperatureCelcius;

		getLogger().debug("");
		getLogger()
				.debug("Pi-Space Transverse Kinetic Energy kelvin
temperature kelvin= "
					+ temperatureKelvin
					+ " mass= "
					+ mass
					+ " to velocity= "
					+
pse.getPiSpaceTransverseKineticEnergySolveForVelocity(
							temperatureKelvin, mass));

		getLogger().debug("");
		getLogger()
				.debug("Newtonian Transverse Kinetic Energy kelvin
temperature kelvin= "
					+ temperatureKelvin
					+ " mass= "
					+ mass
					+ " to velocity= "
					+
pse.getNewtonianTransverseKineticEnergySolveForVelocity(
							temperatureKelvin, mass));

		double accuracy = 0.1;

		assertEquals("Passed",
				pse.getPiSpaceTransverseKineticEnergySolveForVelocity(
						temperatureKelvin, mass),
				pse.getNewtonianTransverseKineticEnergySolveForVelocity(
							temperatureKelvin, mass),
				accuracy);
	}
```

Test Venturi Meter Flow

```
	@Test
	public void test() {

		// Pitot Pressure
		getLogger().debug("");
		getLogger().debug("****** Venturi Meter For Flow Rate");

		PiSpaceFormulas pse = new PiSpaceFormulas(
				PiSpaceFormulas.UNIT_METRES_PER_SECOND);

		double dynamicPressure = 1.0E5; // 1 kilopascal 10^5 newtons
		double density = 820.0; // kerosene

		double diameter1 = 0.1;  //meters
		double diameter2 = 0.06;

		getLogger().debug("");
		getLogger()
				.debug("Venturi Meter For Flow Rate Newtonian pressure to
velocity for dyn pressure= "
					+ dynamicPressure
					+ " density= "
					+ density
					+ " to quantity "
					+ pse.getVenturiMeterForFlowRate(
							dynamicPressure, density,
							diameter1,diameter2));

		getLogger().debug("");
		getLogger()
				.debug("Venturi Meter For Flow Rate Pi-Space pressure to
velocity for dyn pressure= "
					+ dynamicPressure
					+ " density= "
					+ density
					+ " to quantity "
					+ pse.getPiSpaceVenturiMeterForFlowRate(
							dynamicPressure, density,
							diameter1,diameter2));

		//TODO : I may need to use BigDecimal here.

		double accuracy = 1.0;

		assertEquals("Passed",
				pse.getVenturiMeterForFlowRate(
						dynamicPressure, density,
						diameter1,diameter2),
					pse.getPiSpaceVenturiMeterForFlowRate(
						dynamicPressure, density,
						diameter1,diameter2),
				accuracy);
	}
```